

# Point Cloud Edge Detection and Template Matching with 1D Gradient Descent for Wall Pose Estimation

Mitchell Galea, Teresa Vidal-Calleja

Centre for Autonomous Systems at the Faculty of Engineering and IT, University of Technology Sydney  
mitchell.d.galea@student.uts.edu.au, teresa.vidalcalleja@uts.edu.au

## Abstract

Mobile manipulation in unstructured construction environments involves a range of complex robotic problems. We address a perception requirement for autonomous brick placement; estimating the pose of a partially built wall to facilitate the placement of the subsequent brick. Our method uses RGB-D data to extract the surface edge points of the wall and classify them as horizontally or vertically aligned. The contribution of this paper encompasses a wall template that encapsulates its surface edge features and a novel 1D gradient descent template matching algorithm for pose estimation. We apply our method in mobile manipulator brick placement, demonstrating its robotic applications. Evaluation methods prove the efficacy of the proposed framework, both quantitatively and qualitatively and using both simulated and real data.

## 1 Introduction

The vision of machines autonomously performing monotonous and strenuous tasks has fundamentally driven robotics research. Manufacturing industries emanate this vision through autonomous systems operating in structured environments to deliver increases in productivity [Balaguer and Abderrahim, 2008]. However, there is a significant contrast in robotic development when comparing the construction and manufacturing industries [Gambao and Balaguer, 2002]. The increased complexity that robotic applications in unstructured environments demands have inhibited the widespread deployment within the construction industry [Feng *et al.*, 2014]. Autonomous mobile manipulation in dynamic environments for construction applications have the potential to revolutionise the industry by concurrently improving productivity and quality of production [Delgado *et al.*, 2019]. Mobile manipulation refers to a manipulator

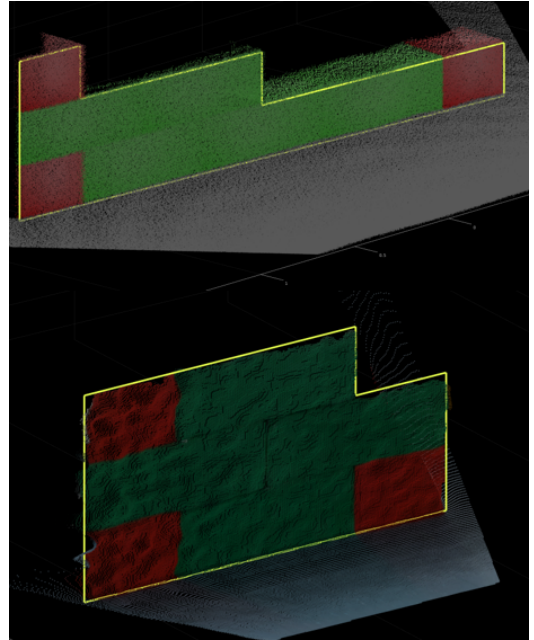


Figure 1: Examples of estimated wall template pose projected onto the point cloud, top is in simulation, bottom is using real data

mounted on a mobile platform [Yamamoto and Xiaoping Yun, 1994]. This paper works towards tackling some of the challenges introduced by the inherently dynamic nature of construction sites that require robust robotic approaches, especially when considering mobile manipulation.

The work in this paper was primarily motivated by the by Mohamed Bin Zayed International Robotics Challenge (MBZIRC) 2020, an international robotics competition that aims to inspire robotic development through the solving of ambitious problems. The challenge requires an Unmanned Ground Vehicle (UGV) to locate, pick, and assemble a set of brick-shaped objects, to construct a digitally pre-specified structure [MBZIRC, 2019]. Mobile manipulator brick placement in unstruc-

tured environments is involved, with various crucial steps in an appropriate robotic pipeline. Accurate brick placement requires the robot to be localised within the environment, perception of the wall and controlled manipulation. The robot also requires an understanding of each bricks relative position within the wall throughout the entire construction process.

This paper addresses the fundamental challenge of perception for the brick placement process. For robotic brick placement an eye-to-hand camera configuration is necessary, eye-in-hand is ruled out due to the brick impeding vision for manipulator mounted cameras. Due to the high precision of the manipulator being used, a single accurate estimation of the target bricks pose, combined with control checks, is sufficient for brick placement. Perception of a wall is not a challenge, however the challenge emanates from perceiving where the subsequent brick placement position is relative to the wall surface.

We address the problem of finding the pose of a brick relative to a wall surface for brick placement by defining a wall template, that encapsulates the surface edges of a wall. We then introduce both a perception approach that utilizes RGB-D point cloud data to detect the wall edges and a novel 1D gradient descent template matching algorithm that estimates the 6DoF pose of the template wall, facilitating the extraction of subsequent target brick poses for accurate placement. The proposed method for wall template pose estimation is evaluated in simulation and with real data. Moreover, we show the proposed template matching methods efficacy through integration with a brick placement pipeline and test the autonomous construction of a wall with a mobile manipulator in simulation.

## 2 Related Work

Model based 3D pose estimation has been widely studied in the literature. Iterative Closest Point (ICP) is the most commonly used algorithm for this application, and involves geometrically aligning two sets of points in which the relative pose is generally close [Besl and McKay, 1992]. However ICP relies heavily on a large correspondence between the two sets of points to reliably converge [Hoda *et al.*, 2015]. This becomes an issue in the desired application of wall pose estimation as usually only a single plane is perceived. This would result in an increased chance of local minima or non convergence. We use a similar method to ICP to minimise the point to template distance in order to converge.

Edge and line detection has been extensively used in computer vision within the RGB domain. The canny line detector is most commonly used for RGB edge detection [CAN, 1987]. Edge detectors provide useful information about structural boundaries of objects in images. Hough

line detection uses a hough transform to extract line features from an image [Duda and Hart, 1972]. Use of edge and line detection in series would allow for detection of wall edges, however pose estimation is more difficult using 2D images in comparison to 3D point clouds. An RGB-D edge detection and registration approach classifies occluding, occluded, boundary, rgb and high curvature edge [Choi *et al.*, 2013].

RGB-D multi plane segmentation approaches aim to segment a point cloud based on planes. Utilising a combination of Random Sample Consensus (RANSAC), Hough Transform and region growing, multi-plane segmentation is possible [Oehler *et al.*, 2011]. Efficient plane detection solves part of the target problem, finding the wall plane, but will not resolve the positioning of the brick within the plane. The segmented planes require further processing to extract the bounding hull for pose estimation of wall templates.

The computation of a region bounded by a series of planar points is of interest for the motivating application. The bounding hull of a series of points can be computed as a convex hull which is the minimal set that will bound the points [Graham and Yao, 1983], or a concave hull which vary depending on variables, for instance the alpha-concave hull which computes the minimum bounding area whilst ensuring that all internal angles are less than  $180 + \alpha$  [Asaedi *et al.*, 2013]. The proposed approach utilizes concave hull to compute the bounding edges of segmented wall planes.

## 3 Notation and Definitions

Let us consider an RGB-D camera and a wall perpendicular to a ground plane. The world frame, camera optical frame, camera footprint frame and wall frame are denoted respectively as,  $\mathfrak{F}_W$ ,  $\mathfrak{F}_C$ ,  $\mathfrak{F}_F$  and  $\mathfrak{F}_{wall}$ . The pose of the camera in the world frame (extrinsic camera parameters) is known and denoted by its homogeneous transformation  $\mathbf{T}_W^C$ .

Note that homogeneous transformation will be used in this paper, hence a given rotation matrix  $\mathbf{R}_a^b$  and translation vector  $\mathbf{p}_a^b$  are associated with  $4 \times 4$  homogeneous transformation matrix from frame a to frame b,u

$$\mathbf{T}_a^b = \begin{bmatrix} \mathbf{R}_a^b & \mathbf{p}_a^b \\ \mathbf{0}^\top & 1 \end{bmatrix} \text{ and } \mathbf{T}_a^{b^{-1}} = \begin{bmatrix} \mathbf{R}_a^b{}^\top & -\mathbf{R}_a^b{}^\top \mathbf{p}_a^b \\ \mathbf{0}^\top & 1 \end{bmatrix}. \quad (1)$$

Let us define the camera footprint frame  $\mathfrak{F}_F$  as the camera shadow on the floor with the  $X$  axis parallel to the ground plane and on coincident plane with  $\mathfrak{F}_C$   $Z$  axis. Hence the translation of  $\mathfrak{F}_F$  is as follows,

$$\text{since } \mathbf{p}_W^C = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \text{ then } \mathbf{p}_W^F = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}. \quad (2)$$

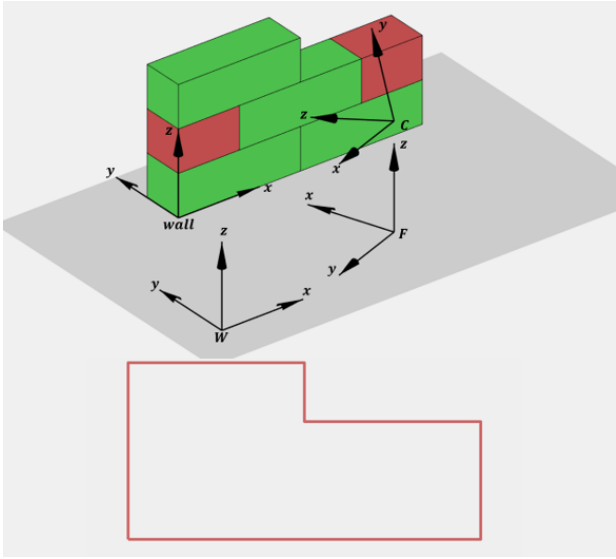


Figure 2: The reference frames that will be used;  $\mathfrak{F}_W$ ,  $\mathfrak{F}_C$  and  $\mathfrak{F}_F$ . The wall reference pose is shown. Below is the wall template for the wall pictured.

Using Euler angles representation to dictate the rotation of the camera  $\mathbf{R}_W^C$ , the rotation matrix  $\mathbf{R}_W^F$  can be found,

$$\text{since } \mathbf{R}_W^C = \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma) \quad (3)$$

$$\mathbf{R}_W^F = \mathbf{R}_z(\alpha). \quad (4)$$

Given  $\mathbf{T}_W^F$ , the pose of the camera footprint with respect to the camera is,

$$\mathbf{T}_C^F = \mathbf{T}_W^C^{-1}\mathbf{T}_W^F. \quad (5)$$

The camera footprint pose is used so that the point cloud given by the depth camera can be transformed, allowing the points  $z$  values to be relative to the ground and the wall plane to be parallel to the camera footprints  $z$  frame, simplifying computation. The reference frames can be seen in Fig. 2.

### 3.1 Wall Template

The wall template is described by the front face edges of a wall. A given wall template is denoted by  $\mathcal{W}$ , and is in reference to the wall reference frame. The wall reference frames'  $X$  axis is aligned with longitudinal direction of the wall,  $Y$  axis is along the depth of the wall and  $Z$  axis is aligned with the height of the wall as per Fig. 2. Note that the wall template poses'  $z$  translation from the world frame is equal to 0, hence,

$$\mathbf{p}_W^{wall} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}. \quad (6)$$

$\mathcal{W}$  consists of an array of lines, which are the edges that make up the wall face. Hence,

$$\mathcal{W} = \left[ \begin{bmatrix} \mathbf{v}_{wall}^{a1} \\ \mathbf{v}_{wall}^{b1} \end{bmatrix} \quad \begin{bmatrix} \mathbf{v}_{wall}^{a2} \\ \mathbf{v}_{wall}^{b2} \end{bmatrix} \quad \cdots \quad \begin{bmatrix} \mathbf{v}_{wall}^{an} \\ \mathbf{v}_{wall}^{bn} \end{bmatrix} \right].$$

Fig.2 shows an example of a wall template.

## 4 Approach

This section details the proposed approach for wall pose estimation using point clouds from RGB-D cameras as input.

### 4.1 Overview

The proposed approach aims to detect and match  $\mathcal{W}$  and estimate the wall's 6DoF pose with respect the camera footprint frame  $\mathbf{T}_F^{wall}$  using a RGB-D camera. Wall edge detection uses dense point cloud data to extract the edge features of a wall face. The template matching algorithm then uses the extracted edge features and a predefined wall template as inputs to 1D gradient descent for 6DoF pose estimation of the wall. 1D gradient descent is possible due to plane detection and translation constraints imposed by the geometry of a wall.

### 4.2 RGB-D Wall Edge Detection

This section details the point cloud processing undertaken to extract wall edge features for template matching. The point cloud processing pipeline assumes input of a dense point cloud generated by an RGB-D camera. The output from the point cloud processing segments horizontal and vertical edge points.

The input is point cloud  $\mathcal{P}_C$ , in the camera reference frame. Every point  $\mathbf{x}_C^i$  in  $\mathcal{P}_C$  is transformed from camera optical frame  $\mathfrak{F}_C$  to the camera footprint frame  $\mathfrak{F}_F$ ,

$$\mathbf{x}_F^i = \mathbf{T}_C^{F-1} \mathbf{x}_C^i.$$

$$\mathcal{P}_C \rightarrow \mathcal{P}_F$$

It is assumed that the wall is within a certain range  $r$  of the camera. In our experiment  $r$  is set to 3m, this is due to the increased noise over this range. Hence the first step of edge detection is to filter  $\mathcal{P}_F$  by  $x$  value. Any point  $\mathbf{x}_F^i$  where  $x_F^i > r$  will be removed from  $\mathcal{P}_F$ . This reduces the impact of noise onto the point cloud and removes possible interference from background objects.

The goal of the point cloud processing is to extract edge points that lie on the front face. We assume that the the camera is facing somewhat towards the wall, more technically the camera  $Z$  axis is more closely aligned to the walls  $Y$  axis than it is to the wall  $X$  axis. Hence the

normal vector to front wall plane  $\mathbf{n}_F = \begin{bmatrix} x_F \\ y_F \\ z_F \end{bmatrix}$  must have

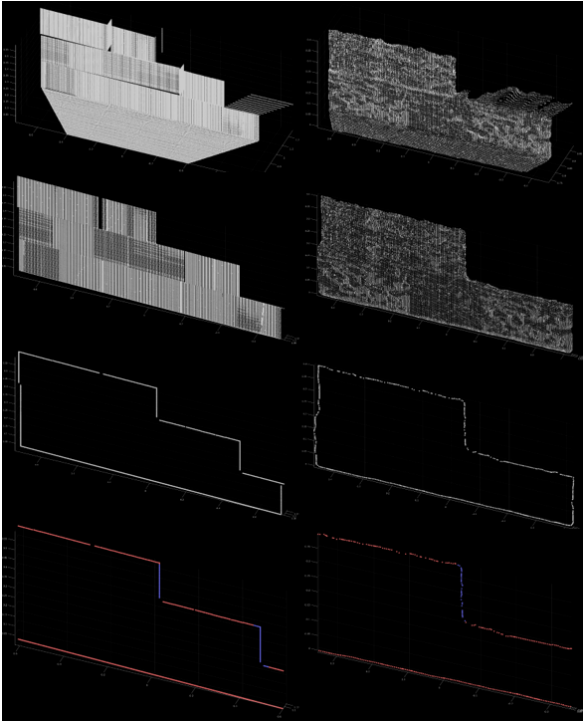


Figure 3: The point cloud processing pipeline. From top, the pre-processed point cloud, the plane detection and projection, the result of the convex hull extraction, the edge detection after field of view filtering and horizontal and vertical point classification. Left is in simulation, right is with realSense D435 RGB-D camera

$x$  and  $y$  values so that,

$$|\tan \frac{y_{CO}}{x_{CO}}| < \frac{\pi}{4}.$$

We use random sample consensus (RANSAC) [Fischler and Bolles, 1981], with a planar model which is constrained to be parallel to a given axis [Alehdaghi *et al.*, 2015]. The axis vector  $\mathbf{a}$  must constrain the plane to be parallel to the camera footprint  $Z$  axis, since the wall is assumed to be perpendicular to the ground plane. hence,

$$\mathbf{a} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

RANSAC extracts the plane coefficients  $\mathbf{c} = [a \ b \ c \ d]$ , where the values relate to the planes equation,

$$ax + by + cz + d = 0$$

Using  $\mathbf{a}$  as a parallel axis constraint causes  $c$  to equal 0. RANSAC finds the plane which contains the most inliers. An inlier is a point  $\mathbf{x}$  in a point cloud that the distance to the plane is less than a variable value  $d^{max}$ . We use

a small  $d^{max}$  value to extract the plane coefficients  $\mathbf{c}$ . It is not beneficial to use a large  $d^{max}$  value because the plane can be shifted away from the wall due to ground plane points and depth wall plane points being inliers.

Once RANSAC is used to extract the coefficients  $\mathbf{c}$  from the point cloud, we use a larger  $d^{max}$  value to extract inliers to the plane with a larger threshold. This is to accommodate for noisy point cloud data and imperfect wall planes. The point to plane distance  $d$  from a point  $\mathbf{x}_0 = (x_0, y_0, z_0)$  to a plane  $ax + by + cz + d = 0$  is given by [Weisstein, 2019a],

$$d = \frac{ax_0 + by_0 + cz_0 + d}{\sqrt{a^2 + b^2 + c^2}}. \quad (7)$$

Hence for every point  $\mathbf{x}_F^i$  in  $\mathcal{P}_F$  if the distance  $d$  to the plane  $\mathbf{c}$ , calculated from Eq. (7), is greater than  $d^{max}$ , the point will be removed.

Points in  $\mathcal{P}_F$  are then projected onto the plane  $\mathbf{c}$  to develop a flat plane  $\mathcal{P}_F^p$ . To do this we first need to generate an arbitrary point  $\mathbf{x}_0 = (x_0, y_0, z_0)$  that lies directly on the plane  $\mathbf{c}$ . Since the plane coefficient  $c = 0$  the plane equation becomes  $ax + by + d = 0$ . We can set  $x_0 = 0$  and  $z_0 = 0$  and solve the equation for  $y_0$ .

The normal of the plane  $\mathbf{n} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ . To calculate the

projection of point  $\mathbf{x}_F^i$  we firstly calculate a vector  $\mathbf{v}$  and take its dot product with the unit normal vector  $\mathbf{n}^u$

$$\mathbf{v} = \begin{bmatrix} x_i - x_0 \\ y_i - y_0 \\ z_i - z_0 \end{bmatrix} \rightarrow d = \mathbf{v} \cdot \mathbf{n}^u.$$

The projected point can now be calculated as,

$$\mathbf{x}_F^{iP} = \mathbf{x}_F^i - d * \mathbf{n}.$$

Now all points in  $\mathcal{P}_F^p$  lie on the plane defined by coefficients  $\mathbf{c}$  [Weisstein, 2019b].

To find the edge features of the point cloud a concave hull can be constructed from the projected point cloud  $\mathcal{P}_F^p$ . A concave hull  $\mathcal{C}_F$  is used rather than a convex hull to allow for internal wall corners to be detected. The “tightness” of the concave hull depends on the input variable  $\alpha$ , a lower value correlates to tighter concave hull however if the value is too low small internal gaps will be extracted as the concave hull [Asaedi *et al.*, 2013].

Once  $\mathcal{C}_F$  is computed, the point cloud will consist of external edges from the wall face. If the camera is positioned in close proximity to the wall plane, the boundary caused by the field of view of the camera may appear as an edge in  $\mathcal{C}_F$ . Since we desire only the wall edges, not point cloud boundaries in  $\mathcal{C}_F$ , we filter the points depending on the horizontal and vertical angle from the cameras origin. The cameras horizontal field of view is





**Data:** Horizontal and Vertical Points  $\mathcal{HP}_F$  and  $\mathcal{VP}_F$   
and Lines  $\mathcal{HL}_{wall}$ ,  $\mathcal{VL}_{wall}$ , wall pose  $\mathbf{T}_F^{wall}$

**Result:** cost  $c$   
 $\mathcal{HL}_F \leftarrow$  Shift H Lines using wall pose  $\mathbf{T}_F^{wall}$ ;  
 $\mathcal{VL}_F \leftarrow$  Shift V Lines using wall pose  $\mathbf{T}_F^{wall}$ ;  
 $c \leftarrow$  Initialize  $c$  to 0;

```

foreach  $\mathbf{x}_F^i$  in  $\mathcal{HP}_F$  do
   $d_{min} \leftarrow$  Initialize to large value;
  foreach  $\mathcal{L}_F^i$  in  $\mathcal{HL}_F$  do
     $d \leftarrow$  calculate distance from  $\mathbf{x}_F^i$  to  $\mathcal{L}_F^i$ ;
    if  $d < d_{min}$  then
       $d_{min} = d$ 
    end
  end
   $c = c + d_{min}^2$ 
end
foreach  $\mathbf{x}_F^i$  in  $\mathcal{VP}_F$  do
   $d_{min} \leftarrow$  Initialize to large value;
  foreach  $\mathcal{L}_F^i$  in  $\mathcal{VL}_F$  do
     $d \leftarrow$  calculate distance from  $\mathbf{x}_F^i$  to  $\mathcal{L}_F^i$ ;
    if  $d < d_{min}$  then
       $d_{min} = d$ 
    end
  end
   $c = c + d_{min}^2$ 
end

```

**Algorithm 1:** Cost Function Algorithm

This is the same process for the vertical points. Using squared distance values allows for more efficient convergence for the gradient descent.

**Data:** Horizontal and Vertical Points  $\mathcal{HP}_F$  and  $\mathcal{VP}_F$   
and Lines  $\mathcal{HL}_{wall}$ ,  $\mathcal{VL}_{wall}$  and iteration count  $it$  and scalar  $s$

**Result:** wall pose  $\mathbf{T}_F^{wall}$  and output cost  $c$   
 $\mathbf{T}_F^{wall}_0 \leftarrow$  Calculate Initial wall pose;

```

foreach  $i$  in  $it$  do
   $\mathbf{T}_F^{wall}_p \leftarrow$  Calculate positive translation pose;
   $\mathbf{T}_F^{wall}_n \leftarrow$  Calculate negative translation pose;
   $c_p \leftarrow$  Calculate positive cost from  $\mathbf{T}_F^{wall}_p$ ;
   $c_n \leftarrow$  Calculate negative cost from  $\mathbf{T}_F^{wall}_n$ ;
   $m \leftarrow$  Calculate gradient,  $c_n - c_p$ ;
   $\mathbf{T}_F^{wall}_i \leftarrow$  Calculate current pose by translating by  $m * s$ ;
end
 $c \leftarrow$  Calculate final cost using final pose  $\mathbf{T}_F^{wall}_i$ 

```

**Algorithm 2:** Template Matching using 1D Gradient Descent

Using the cost function we can now use gradient descent to estimate the pose of the wall, using Alg. 2. We initially have an input wall pose  $\mathbf{T}_F^{wall}_0$ . The gradient descent template matching process will iterate  $it$  times, each time calculating the gradient of the cost function and translating the pose  $\mathbf{T}_F^{wall}$  along the  $X$  axis. For each iteration  $i$  we use the previous pose estimation  $\mathbf{T}_F^{wall}_{i-1}$  to calculate the gradient. We do this by

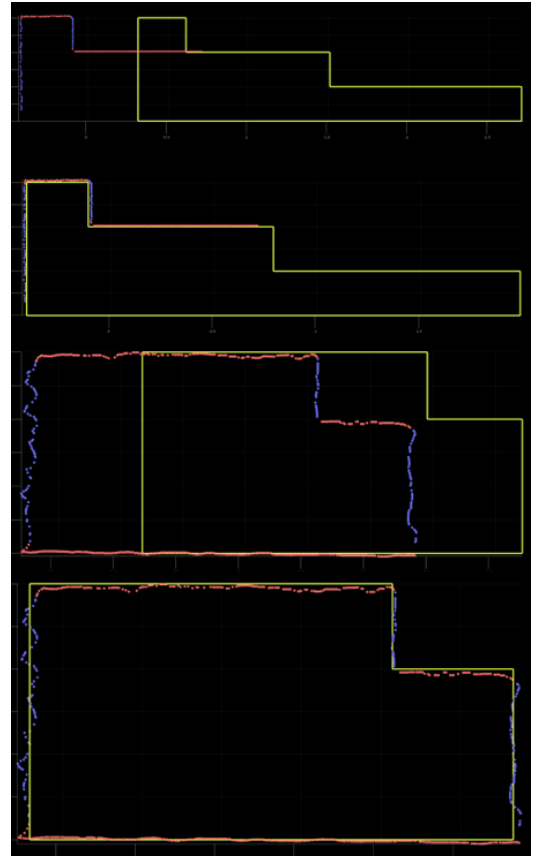


Figure 5: The gradient descent template matching process. Top 2 images were produced in simulation, bottom 2 images using real data. The yellow is the template, red horizontal points and blue vertical points.

translating the pose  $\delta$  distance (0.0001 mm in our implementation) positively and negatively along the  $X$  axis to gain  $\mathbf{T}_F^{wall}_p$  and  $\mathbf{T}_F^{wall}_n$  respectively. We do this by using a translation matrix  $\mathbf{T}_x(l)$  which corresponds to translating  $l$  distance along the poses  $x$  direction. Hence  $\mathbf{T}_F^{wall}_p$  and  $\mathbf{T}_F^{wall}_n$  can be derived by,

$$\mathbf{T}_F^{wall}_p = \mathbf{T}_F^{wall}_{i-1} \mathbf{T}_x(\delta)$$

$$\mathbf{T}_F^{wall}_n = \mathbf{T}_F^{wall}_{i-1} \mathbf{T}_x(-\delta)$$

For both  $\mathbf{T}_F^{wall}_p$  and  $\mathbf{T}_F^{wall}_n$  we calculate the shifted lines and calculate the cost  $c_p$  and  $c_n$  respectively. The gradient is then calculated by  $m = c_n - c_p$ . A positive gradient will mean that the positive pose is better aligned to the points whilst a negative gradient is the opposite. The new pose is calculated by,

$$\mathbf{T}_F^{wall}_i = \mathbf{T}_F^{wall}_{i-1} \mathbf{T}_x(m * s)$$

$s$  is a scalar value that can be adjusted. As the pose approaches the correct value the magnitude of  $m$  will

decrease, causing the translation to decrease, this will allow for accurate positioning. The alignment process can be seen in Fig.5.

## 5 Automated Brick Placement

This section presents an application of the proposed wall template matching framework explained above for autonomous brick wall construction given a pre-specified wall blueprint. The system hardware consists of a robot platform with a mounted serial-link manipulator and RGB-D Camera for perception. The pipeline runs on a node that receives requests from the robot state machine, sending the optimal robot position for placement, brick specification, end effector pose for placement, and finally the completion of the wall structure.

Firstly, the initial request is received; the node will send the optimal robot placement position, and brick specification for the next brick to the state machine. The robot will collect the brick and navigate towards the stated placement pose. From there, when the subsequent request is received; the node will detect and estimate the pose of the wall, using the template matching framework, which enables the calculation of manipulator end-effector pose for brick placement. The node repeats the process above until the completion of the wall structure. Fig. 6 illustrates a flowchart of this pipeline.

### 5.1 Wall Blueprint and Template Extraction

The wall blueprint  $\mathcal{B}$  is a data structure used in the brick placement pipeline that stores the 6DoF pose, colour and placement sequence of each brick in the wall. The wall blueprint data structure has been developed to satisfy the criteria listed in the MBZIRC challenge<sup>1</sup>, which was the primary motivation for this application. The poses stored in  $\mathcal{B}$  for each brick is the centre position of the brick relative to the wall reference pose, which is a corner point on the bottom of the wall. Fig. 7 illustrates this. Hence,

$$\mathcal{B} = [\mathbf{T}_{wall}^{b1} \quad \mathbf{T}_{wall}^{b2} \quad \dots \quad \mathbf{T}_{wall}^{bn}] , \quad (8)$$

where  $\mathbf{T}_{wall}^{bi}$  represents the pose of brick  $i$  in the wall reference frame. The wall reference pose  $\mathbf{T}_W^{wall}$  is in the world frame which allows for the pose of each brick in the wall to be calculated,

$$\mathbf{T}_W^{bi} = \mathbf{T}_W^{wall} \mathbf{T}_{wall}^{bi} . \quad (9)$$

The order that the brick poses are stored in  $\mathcal{B}$  is determined by the sequence of placement. Hence the first brick will be the first brick to place, etc. The poses are not ordered based on position. The ordered wall

blueprint allows for extraction of wall templates for template matching progressively as the wall is built, so that the template changes as bricks are added to the wall. Storing the poses of the bricks also allows for geometric information such as the row and column of the brick in the wall simple to extract.

### 5.2 Robot Position for Brick Placement

The robot position for brick placement  $\mathbf{T}_W^p$  is calculated from the wall blueprint. The ideal placement position for the  $i^{th}$  brick is offset from the centre of the brick. This allows for maximum reach of the manipulator for placement. Also since its a pose for the robot, the  $X$  axis must be pointing towards the brick. To transform from the target brick pose to the placement pose  $\mathbf{T}_{bi}^p$  we must translate along the bricks  $Y$  axis negatively by the offset value  $o$ , then rotate around the  $Z$  axis by 90 degrees to face the wall. Hence,

$$\mathbf{T}_{bi}^p = \begin{bmatrix} \mathbf{R}_z(\frac{\pi}{2}) & \mathbf{p} \\ \mathbf{0}^T & 1 \end{bmatrix} \text{ where } \mathbf{p} = \begin{bmatrix} 0 \\ -o \\ 0 \end{bmatrix}$$

Therefore the robot position for placement for the  $i^{th}$  brick, with an offset  $o$  from the wall is,

$$\mathbf{T}_W^p = \mathbf{T}_W^{wall} \mathbf{T}_{wall}^{bi} \mathbf{T}_{bi}^p$$

### 5.3 Brick Pose Estimation using Template Matching

The robot will travel to the robot place pose  $\mathbf{T}_W^p$ . The RGB-D wall edge detection and template matching is then run to gain the pose of the wall in the robots frame  $\mathbf{T}_R^{wall}$ . The bricks placement position can then be found,

$$\mathbf{T}_R^{bi} = \mathbf{T}_R^{wall} \mathbf{T}_{wall}^{bi}$$

This pose is used to calculate the end-effector position and facilitates brick placement.

## 6 Experiments and Results

This section presents results where the proposed wall pose estimation framework is evaluated quantitatively in simulation and qualitatively using real data. Furthermore, the brick placement pipeline is evaluated in simulation. The pipeline was implemented in C++, using Robot Operating System (ROS) middleware. Point Cloud Library (PCL) was used extensively.

### 6.1 Wall Pose Estimation Simulation

The wall detection and template matching for pose estimation framework has been evaluated using simulated data. The framework has been tested for three difference wall templates, at a near and far vantage point. The near vantage point is 0.7m from the wall, at this

<sup>1</sup><http://www.mbzirc.com/challenge/2020>

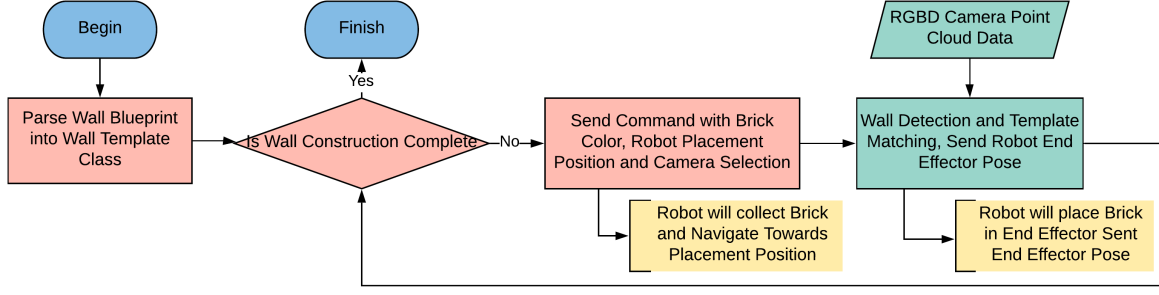


Figure 6: The pipeline flowchart of the proposed method. The input is a wall template and RGB-D Point Cloud Data, outputs are brick color, robot placement position, end-effector brick placement pose.

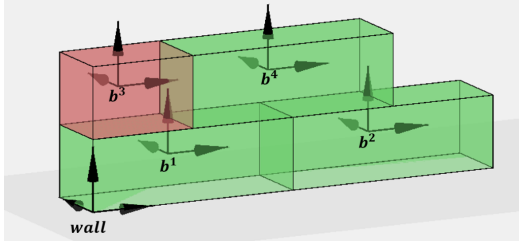


Figure 7: The wall blueprint, the poses of the bricks are centred and relative to the wall reference pose.

distance the field of view limits the visible features. The proposed framework has been tested against the PCL implementation of ICP [Besl and McKay, 1992; Rusinkiewicz and Levoy, 2001]. 3D models of the 3 wall templates were created and mesh sampling [Terzopoulos and Vasilescu, 1991] was used to generate dense point clouds for each respective 3D model. ICP matches a point cloud to another and extracts the transformation.

Due to the  $z$  position and both the roll and pitch of the wall being constrained and assumed constant, with the wall placed on and perpendicular to the ground plane, we have measured the average  $x$ ,  $y$  and yaw errors for both the proposed framework and ICP. The tests have been done with varying levels of simulated noise. To simulate the noise of an RGB-D camera, which isn't typically gaussian, we have added a *sin* wave noise. A scalar value  $s$  is calculated for each points vector and the then multiplied to scale it along its axis. The noise equation for point  $\mathbf{x}_C^i$ , where  $m$  is the magnitude of the vector  $\mathbf{v}_C^i$  from the camera to the point,

$$d \sim \mathcal{N}(\mu, \sigma^2).$$

$$s = \frac{m + 0.5\sigma \sin(20x_C^i) \sin(10y_C^i) + 0.5d}{m}.$$

$$\mathbf{v}_C^i = s\mathbf{v}_C^i.$$

		Template Matching			ICP		
Pos	$\sigma$	x	y	yaw	x	y	yaw
Far	0	3.6	1.5	0.0016	23.1	8.8	0.0014
Far	10	4.6	4.6	0.0033	14.1	9.9	0.0018
Far	20	5.6	8.0	0.0068	27.9	16.4	0.0047
Far	50	7.8	18.7	0.0152	38.0	48.6	0.0160
Near	0	0.4	0.6	0.0013	n.a	n.a	n.a
Near	10	4.3	3.0	0.0091	n.a	n.a	n.a
Near	20	4.2	4.4	0.0122	n.a	n.a	n.a
Near	50	5.8	12.9	0.0274	n.a	n.a	n.a

Table 1: Results for Template Matching in comparison to ICP for Wall Pose Estimation, showing  $x$ ,  $y$  and yaw error.  $x$  and  $y$  and  $\sigma$  values in mm, yaw in radians.

The noisy point clouds can be seen in Figure 9. The results can be seen in Table 1. Comparing the far distance results, where the entire wall is visible, the proposed template matching algorithm performed better than ICP, the average  $x$  error was 22% of ICP, the average  $y$  error was 36% of ICP. The yaw error was similar to ICP. The template matching framework performed better due to its specificity. The performance of ICP was limited by the point clouds showing mainly one face of the wall. The processing time was comparable for both ICP and the proposed framework at approximately 2 seconds, depending on the size of the input point cloud.

For the near wall results ICP performs poorly. ICP can be used to match an object point cloud template to a scene, however if the view is limited ICP fails, the template matching framework extracts the features and matches the template to the features based on the features distance to the template not the template distance to the features. This means that the framework is robust to limited field of view applications. The near view results for the proposed framework is also more accurate than the far results, this is likely due to higher accuracy plane detection.



x	y	yaw
14.0	5.6	0.0171

Table 2: Average brick placement error. x and y values in mm, yaw in radians.

For both near and far tests, with noise ranging from 20mm - 50 mm, the proposed framework reliably produced results accurate to 10mm, and less than 5mm when noise was below 20mm. For the stated application, x and y accuracy was aimed to be below 5mm and yaw accuracy to within a degree, the simulated test results were below the stated ranges when noise was below 20mm.

## 6.2 Wall Pose Estimation Real Data

We have used the Realsense D435 RGB-D camera for collection of real data. The camera is mounted on a robot platform with known extrinsic parameters. The robot has been located in various positions around 2 wall configurations. We have ran the template matching pose estimation framework at the different positions to extract the wall pose in the camera footprint frame. Using the camera extrinsic parameters we are able to project the estimated pose template onto the camera image to gain a qualitative evaluation of the template matching process. Fig. 10 shows the projected templates.

## 6.3 Brick Placement Pipeline Simulation

The brick placement pipeline has been tested on ROS using Gazebo simulator. This can be seen in Fig.8. The resulting error (Table 6.3) is a combination of error from picking the brick, estimating wall pose and manipulator placement. These results validate the performance requirements for mobile manipulation brick placement.

## 7 Conclusion

We presented a model for a wall template, and a template matching method that uses 1D gradient descent to match extracted wall edge features to a given template.

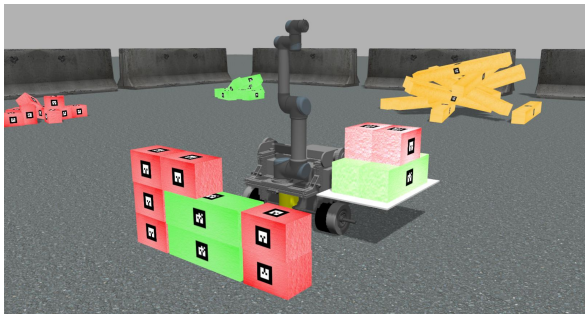


Figure 8: The simulated mobile manipulator in the process of building a wall

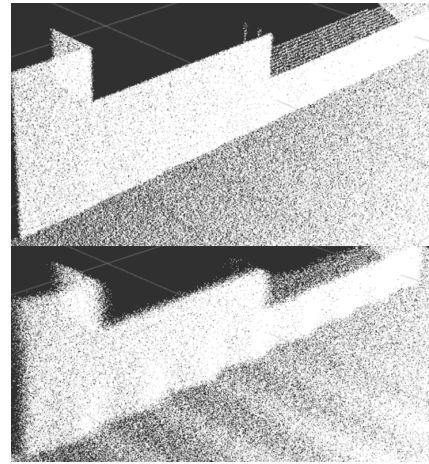


Figure 9: The simulated noise used for testing, from top, 10mm, 20mm 50mm std deviation

Our presented method solves a fundamental perception problem for mobile manipulator brick placement by estimating the pose of a semi-constructed wall to find the pose of the brick to be placed. It was found that the proposed method out performed ICP for estimating the pose of a known wall using RGB-D data in simulation. These results were supported by the successful implementation of brick construction using the template matching process.

## References

- [Alehdaghi *et al.*, 2015] M. Alehdaghi, M. A. Esfahani, and A. Harati. Parallel ransac: Speeding up plane extraction in rgb-d image sequences using gpu. In *2015 5th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 295–300, Oct 2015.
- [Asaeedi *et al.*, 2013] Saeed Asaeedi, Farzad Didehvar, and Ali Mohades. Alpha convex hull, a generalization of convex hull. *CoRR*, abs/1309.7829, 2013.
- [Balaguer and Abderrahim, 2008] Carlos Balaguer and Mohamed Abderrahim. *Trends in Robotics and Automation in Construction*. 10 2008.
- [Besl and McKay, 1992] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992.
- [CAN, 1987] A computational approach to edge detection. In Martin A. Fischler and Oscar Firschein, editors, *Readings in Computer Vision*, pages 184 – 203. Morgan Kaufmann, San Francisco (CA), 1987.
- [Choi *et al.*, 2013] C. Choi, A. J. B. Trevor, and H. I. Christensen. Rgb-d edge detection and edge-based

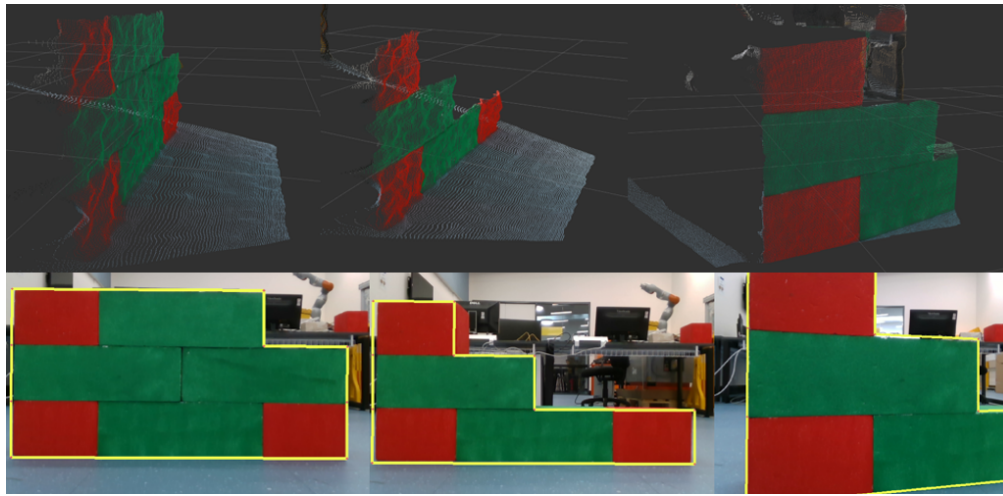


Figure 10: Three test cases showing the input point cloud and the projected estimated wall template onto the image, using RealSense D435 RGB-D camera.

- registration. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1568–1575, Nov 2013.
- [Delgado *et al.*, 2019] Juan Manuel Davila Delgado, Lukumon Oyedele, Anuoluwapo Ajayi, Lukman Akanbi, Olugbenga Akinade, Muhammad Bilal, and Hakeem Owolabi. Robotics and automated systems in construction: Understanding industry-specific challenges for adoption. *Journal of Building Engineering*, 26:100868, 2019.
- [Duda and Hart, 1972] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, January 1972.
- [Feng *et al.*, 2014] Chen Feng, Yong Xiao, Aaron Willette, Wes McGee, and Vineet Kamat. Towards autonomous robotic in-situ assembly on unstructured construction sites using monocular vision. 07 2014.
- [Fischler and Bolles, 1981] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
- [Gambao and Balaguer, 2002] E. Gambao and C. Balaguer. Robotics and automation in construction [guest editors]. *IEEE Robotics Automation Magazine*, 9(1):4–6, March 2002.
- [Graham and Yao, 1983] Ronald L. Graham and F. Frances Yao. Finding the convex hull of a simple polygon. *Journal of Algorithms*, 4(4):324 – 331, 1983.
- [Hoda *et al.*, 2015] T. Hoda, X. Zabulis, M. Lourakis, . Obdrlek, and J. Matas. Detection and fine 3d pose estimation of texture-less objects in rgb-d images. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4421–4428, Sep. 2015.
- [MBZIRC, 2019] MBZIRC. Mbzirc 2020 the challenge, 2019.
- [Oehler *et al.*, 2011] Bastian Oehler, Joerg Stueckler, Jochen Welle, Dirk Schulz, and Sven Behnke. Efficient multi-resolution plane segmentation of 3d point clouds. In Sabina Jeschke, Honghai Liu, and Daniel Schilberg, editors, *Intelligent Robotics and Applications*, pages 145–156, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [Rusinkiewicz and Levoy, 2001] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, May 2001.
- [Terzopoulos and Vasilescu, 1991] Demetri Terzopoulos and Manuela Vasilescu. Sampling and reconstruction with adaptive meshes. *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 70–75, 1991.
- [Weisstein, 2019a] Eric W Weisstein. Point-plane distance, 2019.
- [Weisstein, 2019b] Eric W Weisstein. Projection, 2019.
- [Yamamoto and Xiaoping Yun, 1994] Y. Yamamoto and Xiaoping Yun. Coordinating locomotion and manipulation of a mobile manipulator. *IEEE Transactions on Automatic Control*, 39(6):1326–1332, June 1994.